Computing Lower Bounds for Online Optimization Problems: Application to the Bin Stretching Problem

Michaël Gabay, Nadia Brauner, Vladimir Kotov

Computing Lower Bounds for Online Optimization Problems: Application to the Bin Stretching Problem

Michaël Gabay[a,*], Nadia Brauner[a], Vladimir Kotov[b]

[a] *Grenoble-INP / UJF-Grenoble 1 / CNRS, G-SCOP UMR5272 Grenoble, F-38031, France*
[b] *Belarusian State University, FPMI DMA department, 4 Nezavisimosti avenue 220030 Minsk Belarus*

Abstract

We use game theory techniques to automatically compute improved lower bounds on the competitive ratio for the bin stretching problem. Using these techniques, we raise the best lower bound for this problem to 19/14. We explain the technique and show that it can be generalized to compute lower bounds for any online or semi-online packing or scheduling problem. We also present a first lower bound, with value 7/6, on the expected competitive ratio of randomized algorithms for the bin stretching problem.

Keywords: Bin Stretching, Online Algorithms, Competitive Analysis, Lower Bounds

1. Introduction

In the semi-online bin stretching problem, we are given a sequence of items defined by their weights $w_i \in [0;1]$. They all have to be packed into m bins with infinite capacities. We know in advance that all the items can be packed into m bins with unit size. The items become available and are packed in the order of the sequence, without any knowledge on the number of remaining items and their processing times except that all items fit into m bins with unit size. The value of a solution is equal to the size of the most stretched bin, which is the maximum between 1 and the size of the largest bin. An algorithm with *stretching factor* c for the semi-online bin stretching problem is an online algorithm which successfully packs into m bins of size c, any sequence of items fitting into m unit sized bins. That is, for any instance I, the algorithms outputs a solution with value at most c. The aim is to find an algorithm having a stretching factor as small as possible.

This problem is equivalent to $P_m|online-list|C_{max}$ where we additionally know that the optimal makespan is smaller than or equal to a given value C. The parameter $online-list$ means that, as soon as a job is presented, all its characteristics are known (its processing time in our case) and this job has to be scheduled before the next job is seen. The reader can refer to Borodin and El-Yaniv [1] for more details about online computation and to Pruhs et al. [2] for online scheduling problems.

The bin stretching problem has been introduced by Azar and Regev [3]. They proposed an algorithm of stretching factor 1.625 and proved that 4/3 is the optimal stretching factor with two bins. Other algorithms with improved stretching factor have then been proposed by Kellerer and Kotov [4] and recently by Gabay et al. [5] who respectively proposed algorithms with stretching factors 11/7 and 26/17. The upper bound on the competitive ratio (the stretching factor) for this problem has been improved while, in the meantime, the best known lower bound remained the same: 4/3. In this paper, we present new lower bounds for this problem, for both deterministic and randomized algorithms. In the following section, we define worst-case competitive analysis and present the classical 4/3 lower bound.

*Corresponding author

1.1. A lower bound

An online algorithm A is *c-competitive* if, for any instance I, A provides a solution with value at most c times greater than the optimal value, *i.e.* $\forall$ instance I, $A(I) \leq c \times OPT(I)$. For the bin stretching problem, this yields $A(I) \leq c$ (we are guaranteed that $OPT(I) = 1$).

Our objective is to improve lower bounds on c for a given problem. Ultimately, the aim is to find the smallest competitive ratio c^* among all online algorithms for the problem. This corresponds to finding the greatest value c^* such that for any online algorithm A, there exists an instance I for which $A(I) \geq c^* \times OPT(I)$. In the following, we present the classical online scheduling lower bound for makespan minimization, adapted to the bin stretching problem.

We consider the problem with 2 bins ($m = 2$) and the two following sequences of items in the input:

$$\pi = \left(\frac{1}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}\right) \qquad \pi' = \left(\frac{1}{3}, \frac{1}{3}, 1\right)$$

Obviously, both of these sequences of items can be packed into two unit sized bins.

We consider a c-competitive deterministic online algorithm A for the bin stretching problem. Algorithm A must pack both of these sequences of items with stretching factor at most c.

Either A packs both of the first two items, of size $\frac{1}{3}, \frac{1}{3}$, in the same bin or in different bins. In the first case, with the sequence π, the smallest bin is filled to at least $4/3$, hence $c \geq \frac{4}{3}$. Otherwise, with sequence π', the smallest bins is filled to at least $4/3$, hence $c \geq \frac{4}{3}$. In both case, $c \geq \frac{4}{3}$. Therefore, the stretching factor of any online algorithm is greater than or equal to $\frac{4}{3}$.

Azar and Regev [3] generalized this bound to any number of bins. This bound, however, has not been improved ever since.

Our aim is to improve this lower bound. Obviously, we cannot work with all possible algorithms and instances. Yet, in order to prove that a lower bound is valid, we need to prove that it is valid for all deterministic algorithms. We remark that on a given input, considering all assignments for all items is the same as considering all algorithms. In the following, we model the problem of finding lower bound as a game and restrain the choices of the adversary (which corresponds to restraining the instances).

1.2. Our contribution

We derive a new worst-case lower bound, with value $19/14 \approx 1.3571$. In order to obtain this bound, we model the problem as a two-players, zero-sum game. Then, we use the so-called adversary method in which a malicious, omnipotent, adversary is playing against the algorithm to derive improved lower bounds. Contrary to traditional studies in online scheduling which often use layering techniques to derive lower bounds for deterministic algorithms, see e.g. [6, 7, 8], we use an automated approach based on the minimax algorithm [9], with alpha-beta pruning [10] to solve the game where the adversary has restricted choices on items weights. Moreover, to comply with the known feasibility of the corresponding bin packing problem with unit sized bins, we use constraint programming to compute feasible decisions of the adversary.

The algorithm outputs a decision tree as a proof. All decisions of the adversary are provided in this tree, for all decisions of any algorithm. The proof for the $19/14$ lower bound is provided in Appendix A.

This computational approach can be generalized and applied to any online or semi-online problem and, to the best of our knowledge, has not been applied yet in the online scheduling literature.

By applying Yao's minimax principle [11], we also obtain a lower bound with value $7/6$ for the expected competitive ratio of any randomized algorithm on the bin stretching problem. The reader can refer to [12] for multiple applications of Yao's principle on scheduling problems.

1.3. Outline

In Section 2, we model the problem of finding lower bounds for bin stretching algorithms as a game. Then, in Section 3, we present the algorithm and cuts we use to solve this game and compute lower bounds. Finally, in Section 4, we present a first lower bound on randomized algorithms for the bin stretching problem.

2. The bin stretching game

We model the problem of finding lower bounds for the bin stretching problem as the following two players, zero-sum, infinite game:

BIN STRETCHING GAME

Player 1 choses a positive integer m. Then, successively, until Player 1 choses `Stop`:

1. Player 1 (the *adversary*) choses a feasible weight defining an item or `Stop`.
2. Player 2 (the *algorithm*) selects an integer $i \in \{1, \ldots, m\}$ and packs the item into the bin B_i.

The payoff of Player 1 is equal to $\max(1, \max_{i=1,\ldots,m} w(B_i))$, where $w(B_i) = \sum_{j \in B_i} w_j$.

Let w_j be the weight selected by Player 1 on iteration j. The weight w_j is feasible if and only if the bin packing problem with m bins of unit capacities and items with weights $w_1, \ldots, w_j$ is feasible. The bin packing problem is strongly $\mathcal{NP}$-hard [13]. However, we can consider that the adversary is an oracle and can easily compute this problem.

Additionally, this is a game with complete information which means that both players know all the decisions taken and recall the history of the game.

The payoff of Player 1 is c, the stretching factor, while the payoff of Player 2 is $-c$. This game is a minimax game where Player 1 aims at maximizing c while Player 2 aims at minimizing c. An algorithm for the bin stretching problem defines a behavior for Player 2. The worst-case competitive ratio of an algorithm is equal to the supremum of c when Player 2 acts according to the algorithm. The supremum on the payoff of Player 1 in this game is equal to the value c^*.

It is easy to see that this game is infinite since the adversary can provide the input $w_j = 1/2^j$, for $j = 1, \ldots, \infty$. Hence, we cannot explore all feasible choices of the adversary unless we restrain them. To cope with this issue, we actually consider that Player 1 has the following behavior: at the beginning of a game, Player 1 choses a positive integer C. Then, all the weights chosen by Player 1 are in $\{1/C, 2/C, \ldots, 1\}$ (he can still chose `Stop` as well). Considering this subset of adversaries, the game is finite: Player 1 has at most mC choices before the game is over.

In order to prove that a value c is a lower bound on c^*, it is "sufficient" to show that for any algorithm, there is an instance such that the stretching factor of the algorithm is greater than or equal to c. We cannot consider all algorithms but, on a given instance, there is a finite number of decisions for Player 2 and considering all decisions is actually the same as considering all algorithms. Hence, we only need to show that, for any decision of Player 2, there is a sequence of decisions from Player 1 leading to a solution with value at least c. Figure 1 illustrates this for the $4/3$ lower bound. All decisions from Player 2 are considered while only one decision for each branch is provided for Player 1.

3. Implementation

In order to solve the game, that is, find a strategy for Player 1, maximizing c, we implement the minimax algorithm (depth-first search) for the game previously described. We apply the alpha-beta pruning with several additional cuts. Remark that considering unit capacities and weights in $\{1/C, 2/C, \ldots, 1\}$ is the same as considering the capacities of the bins to be C and weights in $\{1, \ldots, C\}$. Hence, we represent an item by an integer in $\{1, \ldots, C\}$ and a bin by a list of integers, corresponding to the items in the bin.

3.1. Decisions on items weights and assignments

In order to decide whether an item can be proposed by the adversary or not, we apply simple lower and upper bounding results on the corresponding bin packing problem, including the additional new item. Some of these are described in the following paragraphs.

Let $w_1, \ldots, w_j$ be the weights of the items, sorted in non-increasing order. We verify that $\sum_{i=1}^{j} w_i \leq mC$ and $w_m + w_{m+1} \leq C$. Let $k = \max\{i | w_i > C/2\}$ ($k = 0$ if there are no such items) and $l = \max\{i | w_i = C/2\}$

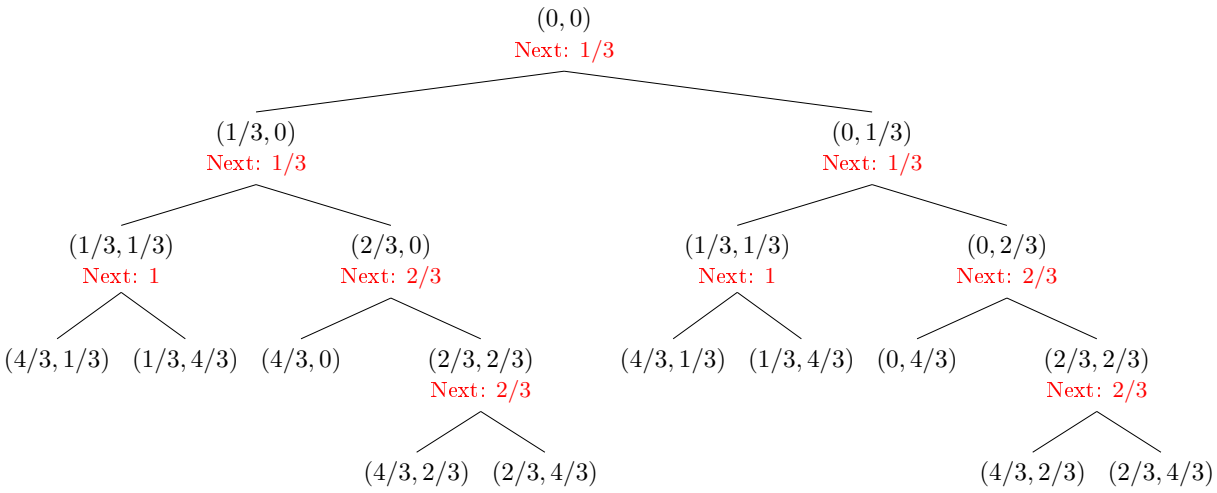


Figure 1: 4/3 lower bound decision tree. Player 1 decisions are the "Next: w_i". The pairs (w_1, w_2) give the space used in the bins.

($l = k$ if there are no such items), we also ensure that $2k + l \leq 2m$. If any of the previous inequalities is not verified, then the weight is infeasible.

Other lower bounds, such as L_2 and L_3 from Martello and Toth [14], can also be computed. However, refined bounds can be computed later, before or during the exact resolution of the problem.

Then, if the problem was not proven infeasible, we compute the best fit decreasing heuristic on the input. If it is feasible, then the new item is accepted. Otherwise, we need an exact approach to determine whether current item is feasible or not.

In our case, we use constraint programming to solve the bin packing problem. This choice was motivated by the small sizes of the problems that have to be solved. We did not implement a dedicated approach since the time spent checking feasibility is strongly dominated by the time spent in the minimax algorithm.

In general, for a semi-online problem, one can use any approach, including integer programming, branch and bound or any exact dedicated approach to determine whether a move for the adversary is feasible or not. We can also use heuristic approaches with the risk of not being able to find a lower bound because of a missed feasible move. However, when a move is validated, it has to be really feasible in order to ensure the correctness of the results of the algorithm.

Eventually, it is not necessary to verify feasibility for all items: once an item is proven feasible, all smaller items are feasible as well. Hence, by considering adversary choices by decreasing order of the weights of the items, we only have to find the first feasible item ; then all remaining items are smaller, hence feasible.

3.2. Cuts

The size of the minimax tree is exponential in m and C. Hence, we have to find a way to cut branches in order to be able to compute optimal solutions. The first step to reduce the minimax tree is to break symmetries on the game: permutation on a the bins are actually corresponding to identical solutions. Moreover, from Player 2 point of view, the items in the bins do not matter. Only the bin sizes matter. So, the configuration $((6, 3), (4, 5), \emptyset)$ is actually the same as $((7, 1, 1), \emptyset, (3, 6))$. However, these two configuration are different from Player 1 point of view since he needs to ensure that the resulting bin packing problem will be feasible. However, to both Player 1 and Player 2, the following configurations are equivalent: $((6, 5, 1, 2), (7, 7), \emptyset)$, $((6, 1, 7), (5, 2, 7), \emptyset)$. These two nodes can actually be described as: $(\{(0, 1), (14, 2)\}, \{(1, 1), (2, 1), (6, 1), (7, 2)\})$ which is the same node to both players. In the first set of pairs, the weights of the bins and their multiplicities are given, while the second set of pairs denotes the weights of the items and their multiplicities. All nodes having the same encoding are equivalent.

We use this encoding to represent a (partial) solution and we take advantage of it in two ways: when Player 2 packs an item, the number of edges to explore is equal to the cardinality of the first set of the pair, which is less than or equal to m. Moreover, we use memoization [15] (and compression) to store and recall the results of the nodes we have already computed and bin packing problems which have already been non-trivially solved. However, because of the pruning, we have to store the values of α and β as well as the evaluation of the node to be able to determine whether the node value shall be recomputed or not when it is recalled.

We apply an alpha-beta pruning to the minimax algorithm. The idea is to maintain a lower bound α and an upper bound β on the stretching factor. The pruning works as follows: on a maximizer node, once it is known that the solution of this node will be better than the solution of another node having the same parent (this parent is a minimizer), it is not necessary to explore any other choice. And similarly for minimizer nodes.

Since the adversary is computing a solution against all algorithms, we can consider several particular algorithms. Especially, we can consider the algorithm packing all remaining items into the currently smallest bin. We do not know the remaining items, but we know that the sum of their weights cannot exceed $mC - \sum_{k=1}^{j} w_k$. Let B_i be the smallest bin, if $w(B_i) + mC - \sum_{k=1}^{j} w_k \leq \alpha$ then we can immediately proceed to a β cut-off.

Additionally, we are aiming at strictly improving known lower bounds and we know that some competitive ratio can be achieved by some deterministic algorithms. So, we start the exploration with a lower bound which is equal to the best known lower bound ($\alpha = \lfloor C\tilde{c} \rfloor$, where $\tilde{c}$ is the best known lower bound on c^*) and an upper bound which is equal to the competitive ratio of the best algorithm ($\beta = 26C/17$).

For most values, the lower bound will not be increased, so we improve this approach by dividing it into two steps: in the first step, we determine whether or not the lower bound can be improved. If so, in a second step, we determine the new best lower bound. Otherwise, we go on to the next value. Thus, we start with $\alpha = \lfloor C\tilde{c} \rfloor$ and $\beta = \lfloor C\tilde{c} \rfloor + 1$; such close values allow very early cut-offs. When the algorithm is over, the value is either α or β. In the first case, the lower bound cannot be improved for current values of m and C. It is over, we can try a new set of parameters m, C. In the latter case, we know that $\frac{\lfloor C\tilde{c} \rfloor + 1}{C}$ is a new, strictly larger lower bound. We re-run the algorithm with $\beta = 26C/17$ to see if we can further improve this new lower bound.

3.3. Results

We implemented the algorithm in `Python` and used Choco [16] as a constraint programming solver. The source code is available online[1].

Running the program with parameters $m = 3$ and $C = 14$, we obtain the lower bound $19/14 \approx 1.357$. We backtracked the results and verified them manually. The proof is provided in Appendix A.

We used `PyPy` interpreter on a computer running Linux and equipped with an Intel Core i7-2600K Processor (clock speed 3.40GHz) and 4GB of RAM to compute lower bounds with our algorithm. Some experimental results are presented Table 2. Using our approach, we were able to compute the results for $m = 3$ and C up to 20, and $m = 4$ and C up to 12. Using the two-step approach, we were able to prove that neither $m = 3$, $C = 21$, nor $m = 4$, $C = 13$ allow to increase the lower bound. With larger values of m, we are only able to compute results with small C and we do not get improved lower bounds. For larger values of C, the number of nodes is too large and we are facing time and memory issues. We did not put much effort into optimizing the code since we are limited by the combinatorial explosion rather than any algorithmic factor.

The results presented Table 2 (except the last column) concern the single step approach, with pruning parameters initialized to $\alpha = \lfloor 4C/3 \rfloor$ and $\beta = 26C/17$. The last column gives the number of nodes in the first stage of the two-steps approach, with $\alpha = \lfloor C\tilde{c} \rfloor$ and $\beta = \lfloor C\tilde{c} \rfloor + 1$.

The column #calls corresponds to the number of times an item feasibility was verified. The column #exact is the number of calls to the exact method (that is when the item was not proven to be feasible or

[1] https://github.com/mgabay/Bin-Stretching-Lower-Bounds

infeasible by a heuristic or a lower bound). The combinatorial explosion is very well illustrated in column #nodes where we can see that even with many efficient cuts, we cannot tackle much larger problems. Table 2 also shows that the time spent verifying items feasibility is negligible compared to the whole time spent. Time spent is approximately linear in the number of nodes, except for the largest instances ($\approx 2\times10^7$ nodes) since the computer is out of memory and swaps, making the algorithms very inefficient.

m	C	c	*feasibility check*			*overall*		*first step*
			#calls	*#exact*	*time*	*#nodes*	*time*	*#nodes*
3	10	—	4,687	37	0.4	49,055	1.4	41,753
	11	—	14,802	141	1.2	168,380	3.4	141,176
	12	—	9,125	63	0.5	118,925	2.2	98,186
	13	—	32,538	209	1.1	458,183	5.8	384,052
	14⋆	**19/14**	82,868	644	2.2	1,240,619	14.0	286,845
	15	—	55,929	344	1.2	890,291	10.1	702,449
	16	—	196,835	1142	3.3	3,384,144	35.5	2,901,483
	17	23/17	207,133	1,804	4.0	3,728,386	40.8	1,620,468
	18	—	303,725	1,646	4.3	5,692,383	57.2	4,652,427
	19	—	1,045,692	4,958	21.0	21,262,246	1225.1	18,653,870
	20	27/20	977,992	6,191	21.9	20,283,070	1046.7	11,446,232
4	7	—	6,622	50	0.4	50,642	1.6	39,946
	8	—	28,099	182	1.1	254,344	4.5	193,474
	9	—	30,991	98	0.8	331,112	4.8	266,926
	10	—	127,063	721	2.6	1,442,281	19.5	1,106,147
	11	—	503,560	3114	7.5	6,365,822	81.7	5,195,618
	12	—	491,497	1974	5.8	6,718,232	89.7	5,158,805
	13	—	1,540,000	>8000	>41.6	>22,900,000	>3600	19,956,339

Figure 2: Numerical results on some inputs. Column 3, c is the best lower bound on the competitive ratio obtained for the instance. "—" means that the 4/3 lower bound was not improved.

In order to improve this algorithm, the first step would be to reduce memory usage by restraining the amount of memoized data. Then, we could use a breadth-first search and for each depth, use a heuristic to select a sample of least promising nodes. Exploring these nodes in depth, will allow some early cut-offs. Another approach is to set $C = 1$ and select random weights in $]0;1]$. Then, we can run the algorithm on many random samples of items, hopefully resulting in an improved lower bound. We ran some tests using this approach but did not obtain any improved lower bound.

4. Lower Bound on randomized algorithms

We are now interested in randomized algorithms for this problem. A randomized algorithm can take its decisions at random, according to some probability distributions. While the previous worst case analysis holds, when designing a random algorithm, the aim is to minimize the expected competitive ratio rather than the worst case competitive ratio. In the following, we prove a lower bound on the expected competitive ratio for any randomized algorithm for this problem.

Theorem 1. *Any randomized algorithm for the semi-online bin stretching problem, has an expected competitive ratio of at least* 7/6*, for any number of machines* $m \geq 2$.

Proof. We use Yao's minimax principle and consider a randomized adversary against a deterministic algorithm. Yao's principle states that a lower bound $\tilde{c}$ for the competitive ratio of deterministic algorithms on a fixed distribution over inputs is also a lower bound for any randomized algorithms.

Let m be the number of machines. We consider the input from Azar and Regev [3], with an additional distribution of probabilities:

- with probability p, the input is m items of weight $1/3$, followed by m item of weight $2/3$.
- with probability $1-p$, the input is m items of weight $1/3$, followed by an item of weight 1.

Both of these inputs are obviously feasible.

Any deterministic algorithm either packs the m first items in different bins or at least two of them are in the same bin. In the first case, the first input yields solutions with value at least 1, while the second input yields solutions with value at least $4/3$. Otherwise, the first input yields solutions with value at least $4/3$, while the second input yields solutions with value at least 1.

Hence, the performance of any deterministic algorithm packing the m first items in different bins is at least $p \times 1 + (1-p) \times 4/3$, while the other deterministic algorithms yield solutions with value at least $p \times 4/3 + (1-p) \times 1$. The min of both of these values is maximized for $p = 1/2$. In such case, the performance of any deterministic algorithm is at least $7/6$ on this input.

Hence, by Yao's principle, $7/6$ is a lower bound on the competitive ratio of any randomized algorithm for the semi-online bin stretching problem. □

5. Conclusion

We improved the best known lower bound for the bin stretching problem and proposed a first lower bound for the performance of randomized algorithms on this problem. The $19/14$ lower bound has been proven for $m = 3$ only, so it is still open to know whether or not $19/14$ is a lower bound for $m \geq 4$. The $7/6$ lower bound for randomized algorithms is valid for all $m \geq 2$.

We represented the problem as a game and used an automated search to find lower bounds. This approach differs from traditional approaches using layering techniques and can be generalized and applied to many other packing or scheduling, online or semi-online problems. Compared to layering techniques, for multiple machines problem, there is however a trade-off on the generality of the bound: the lower bound cannot easily be generalized to any number of machines.

Acknowledgments

The research of the first and second authors has been partially supported by the LabEx PERSYVAL-Lab (ANR–11-LABX-0025) and by BRFFR-PICS project (PICS 5379). The research of the third author has been partially supported by project ICS No 5379 and Belarusian BRFFI grant (Project F13K-078).

References

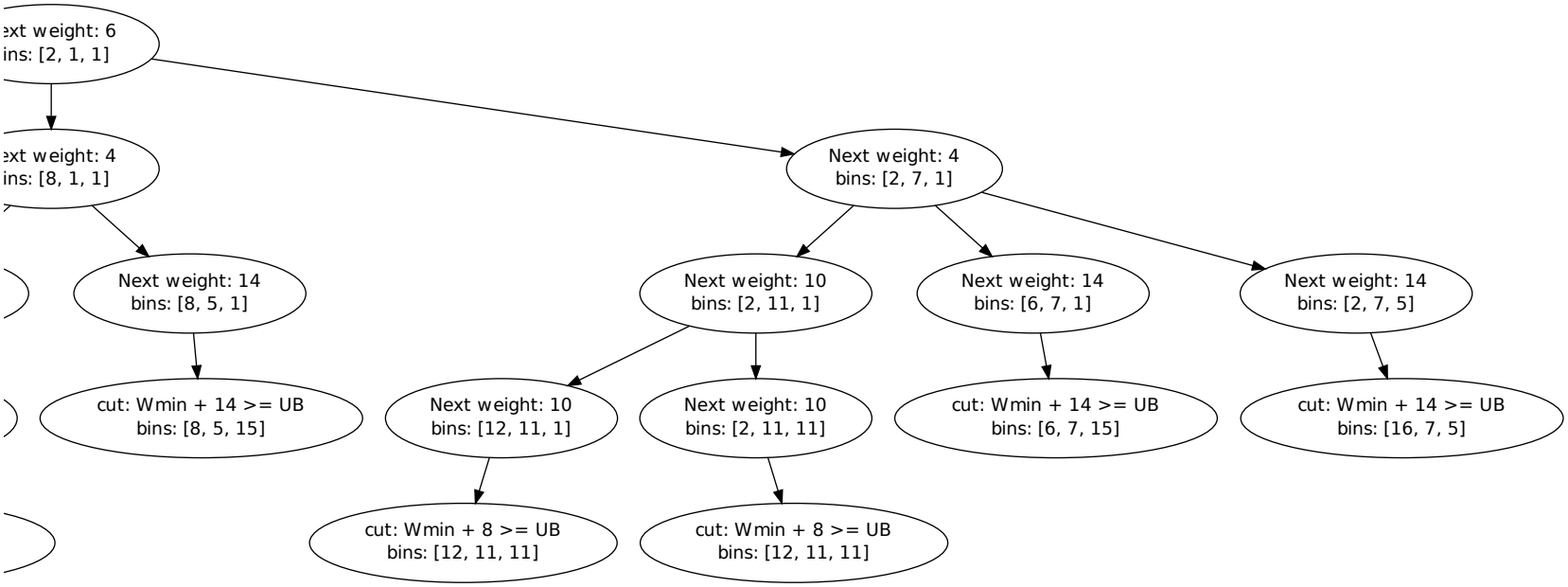
[1] A. Borodin, R. El-Yaniv, Online computation and competitive analysis, vol. 53, Cambridge University Press, 1998.
[2] K. Pruhs, J. Sgall, E. Torng, Online scheduling, in: J. Y. Leung (Ed.), Handbook of scheduling: algorithms, models, and performance analysis, CRC Press, 2004.
[3] Y. Azar, O. Regev, On-line bin-stretching, Theoretical Computer Science 268 (1) (2001) 17–41.
[4] H. Kellerer, V. Kotov, An efficient algorithm for bin stretching, Operations Research Letters 41 (4) (2013) 343–346.
[5] M. Gabay, V. Kotov, N. Brauner, Semi-Online Bin Stretching with Bunch Techniques, Les Cahiers Leibniz 208 (2013) 1–10.
[6] Y. Bartal, H. Karloff, Y. Rabani, A better lower bound for on-line scheduling, Information Processing Letters 50 (3) (1994) 113–116.
[7] S. Albers, Better bounds for online scheduling, SIAM Journal on Computing 29 (2) (1999) 459–473.
[8] J. Rudin, R. Chandrasekaran, Improved Bounds for the Online Scheduling Problem, SIAM Journal on Computing 32 (3) (2003) 717–735.
[9] J. v. Neumann, Zur theorie der gesellschaftsspiele, Mathematische Annalen 100 (1) (1928) 295–320.
[10] J. Pearl, The solution for the branching factor of the alpha-beta pruning algorithm and its optimality, Communications of the ACM 25 (8) (1982) 559–564.
[11] A. C.-C. Yao, Probabilistic computations: Towards a unified measure of complexity, in: 18th Annual Symposium on Foundations of Computer Science, 222–227, 1977.

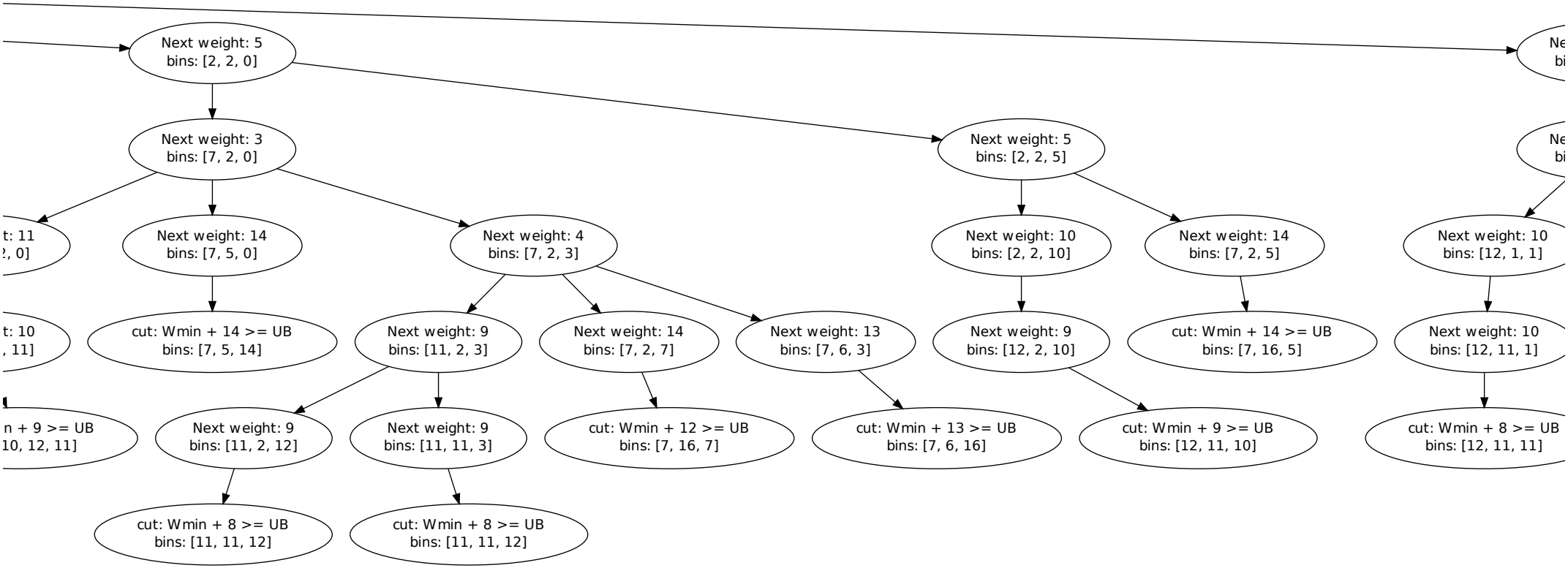
[12] L. Epstein, R. van Stee, Lower bounds for on-line single-machine scheduling, Theoretical Computer Science 299 (1) (2003) 439–450.
[13] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, WH Freeman and Company, New York, 1979.
[14] S. Martello, P. Toth, Lower bounds and reduction procedures for the bin packing problem, Discrete Applied Mathematics 28 (1) (1990) 59–70.
[15] D. Michie, Memo functions and machine learning, Nature 218 (5136) (1968) 19–22.
[16] N. Jussien, G. Rochart, X. Lorca, et al., Choco: an open source java constraint programming library, in: CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08), 1–10, 2008.

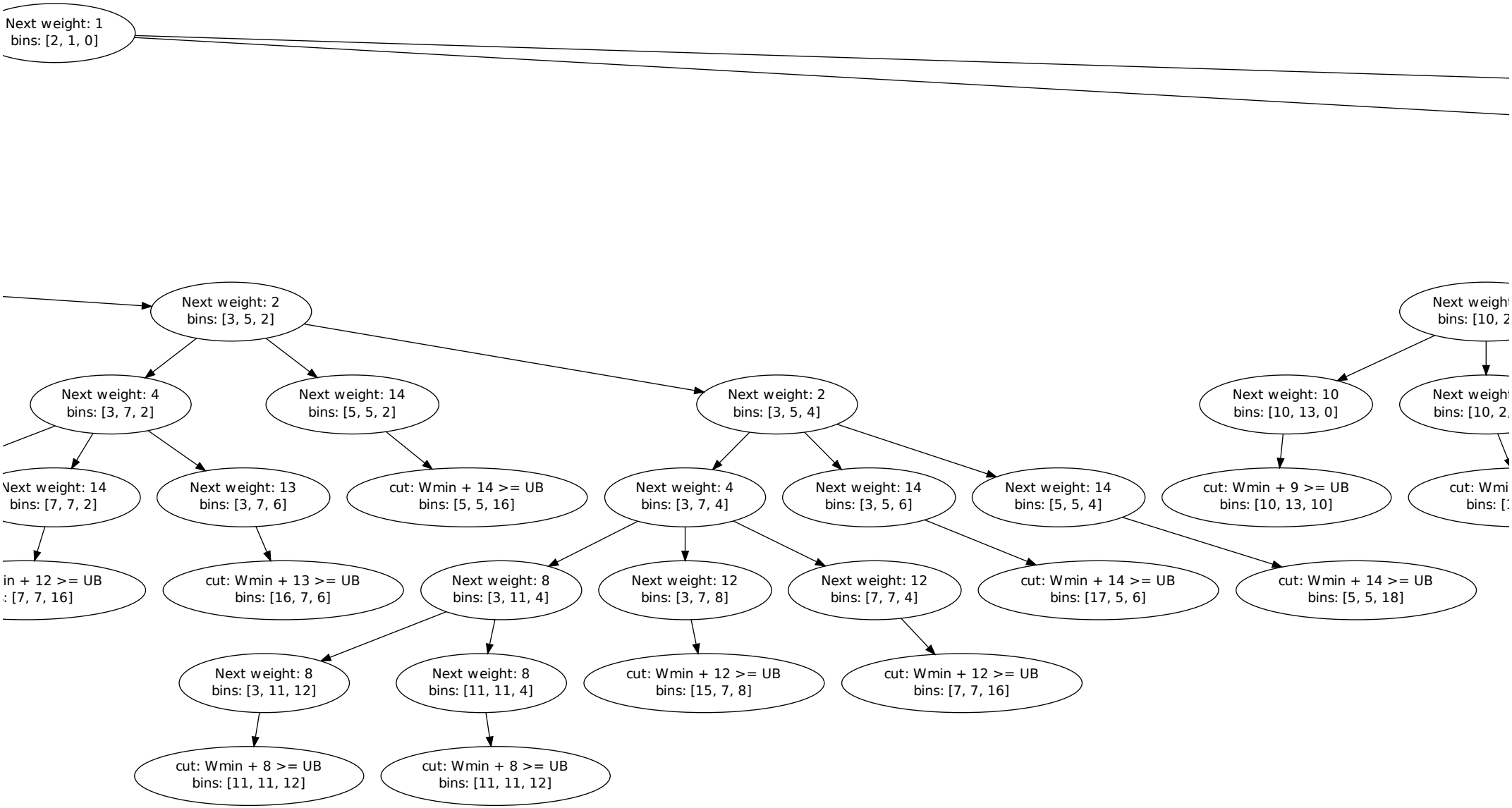
Appendix A. Bin stretching lower bound

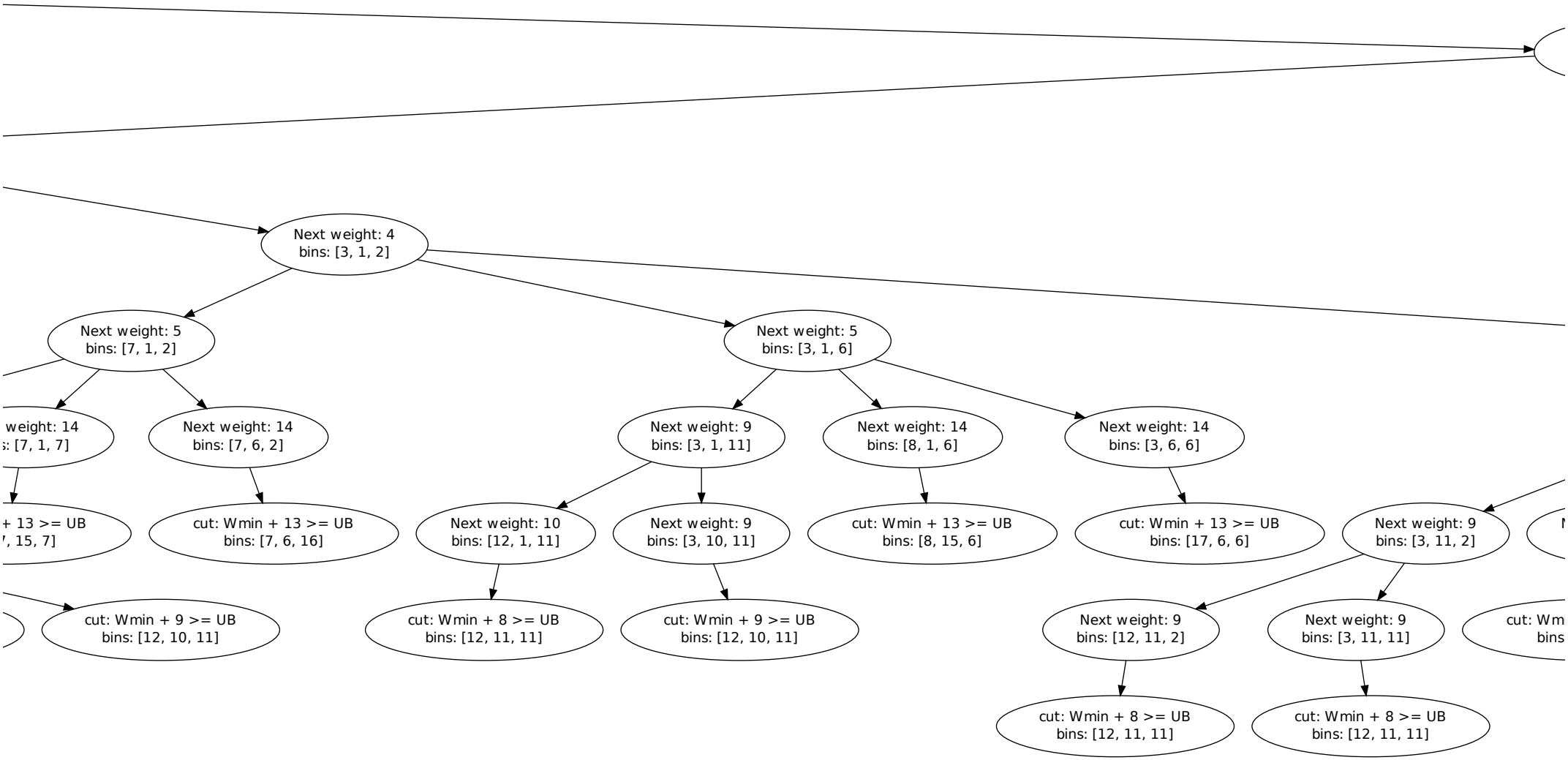
The following tree proves the 19/14 lower bound for the bin stretching problem. This lower bound was obtained using our algorithm with parameters $m = 3$ and $C = 14$.

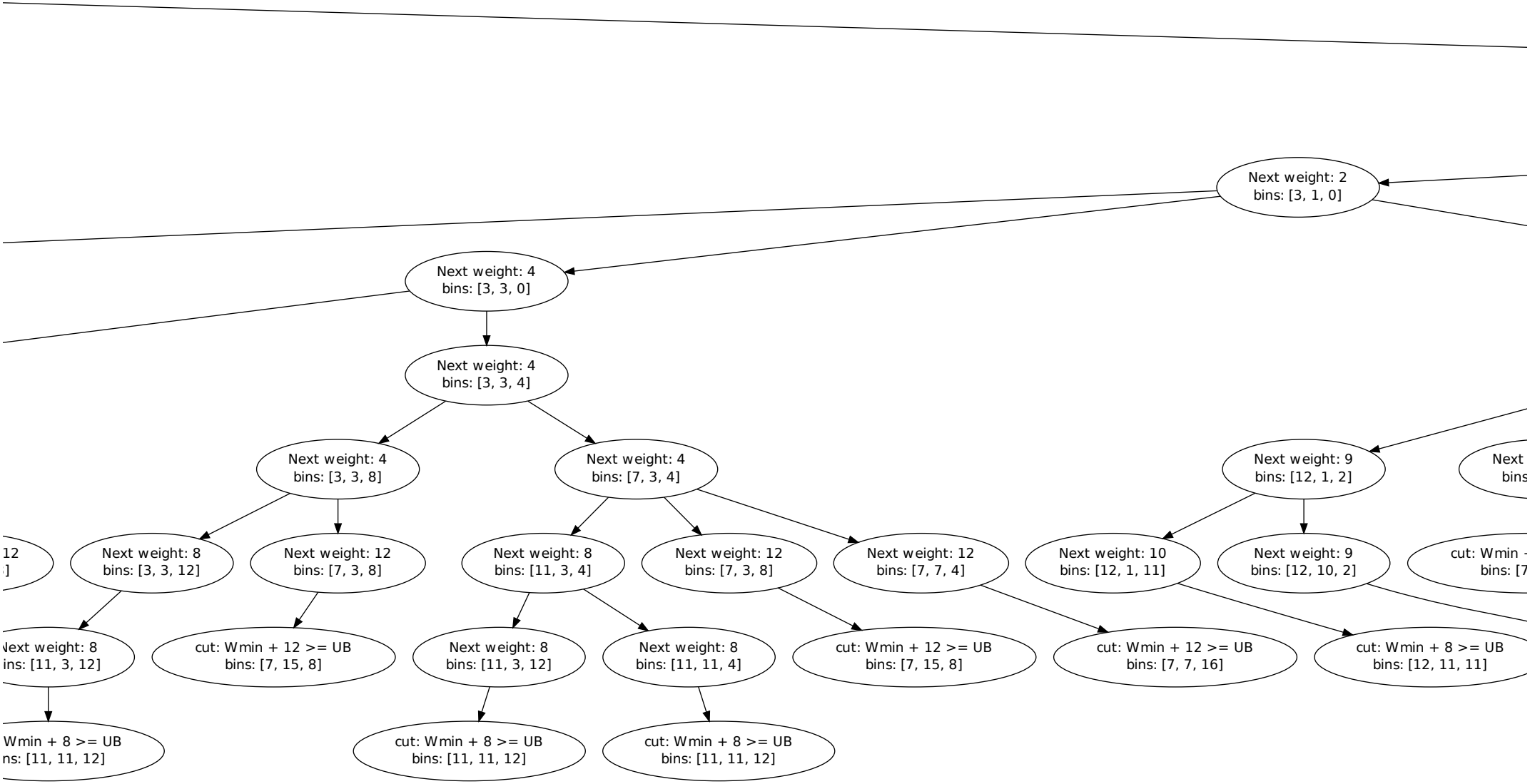
In the proof, a single decision of the adversary is provided for each decision of the algorithm. We do not explore branches where the algorithm packs the item in a bin, making it larger than or equal to 19. As soon as there exists a feasible item such as all algorithms fail, we stop exploring as well. We denote these latter nodes by "Cut: Wmin+w_j>=UB".

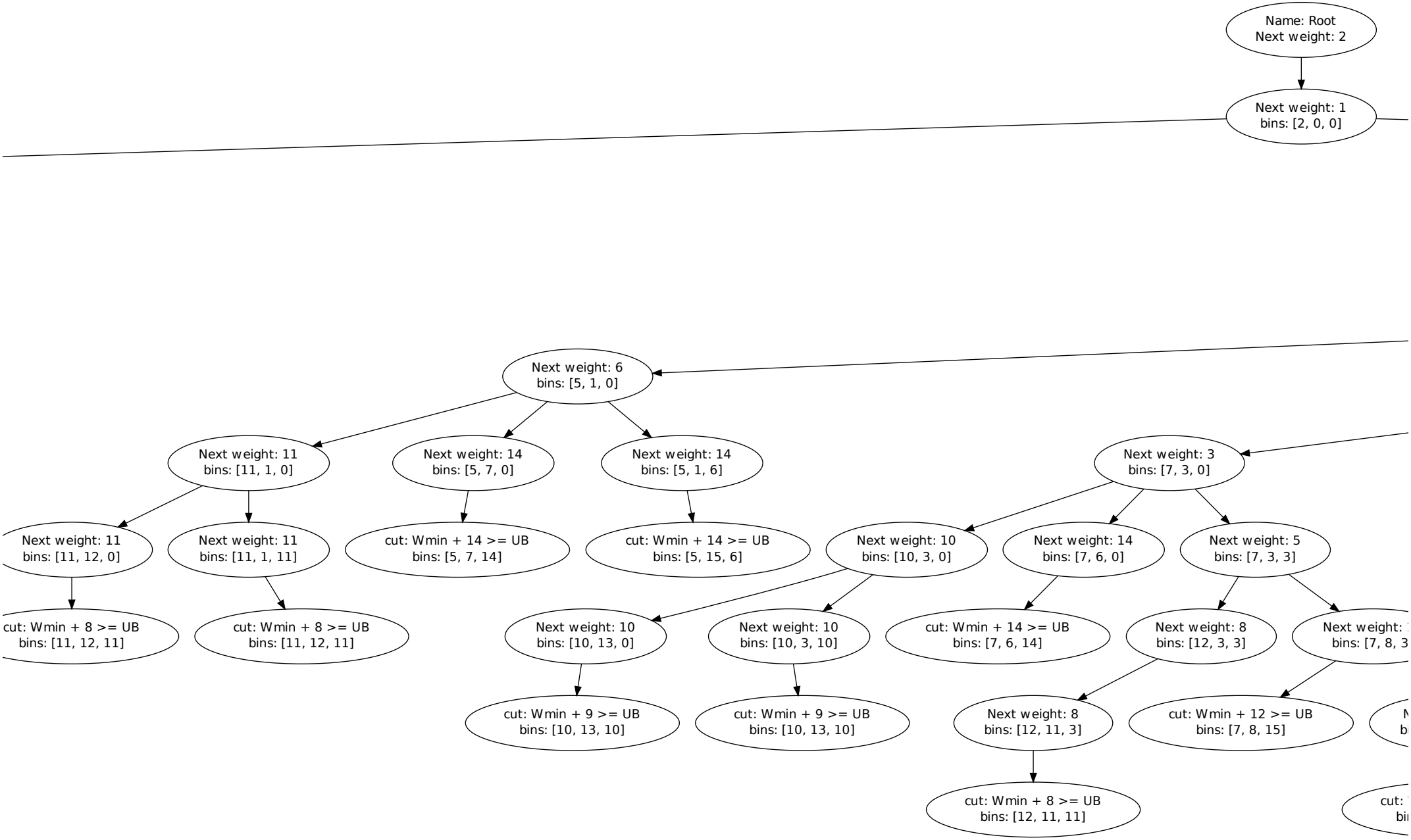

ext weight: 6
ins: [2, 1, 1]
ext weight: 4
ins: [8, 1, 1]
Next weight: 4
bins: [2, 7, 1]
Next weight: 14
bins: [8, 5, 1]
Next weight: 10
bins: [2, 11, 1]
Next weight: 14
bins: [6, 7, 1]
Next weight: 14
bins: [2, 7, 5]
cut: Wmin + 14 >= UB
bins: [8, 5, 15]
Next weight: 10
bins: [12, 11, 1]
Next weight: 10
bins: [2, 11, 11]
cut: Wmin + 14 >= UB
bins: [6, 7, 15]
cut: Wmin + 14 >= UB
bins: [16, 7, 5]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]

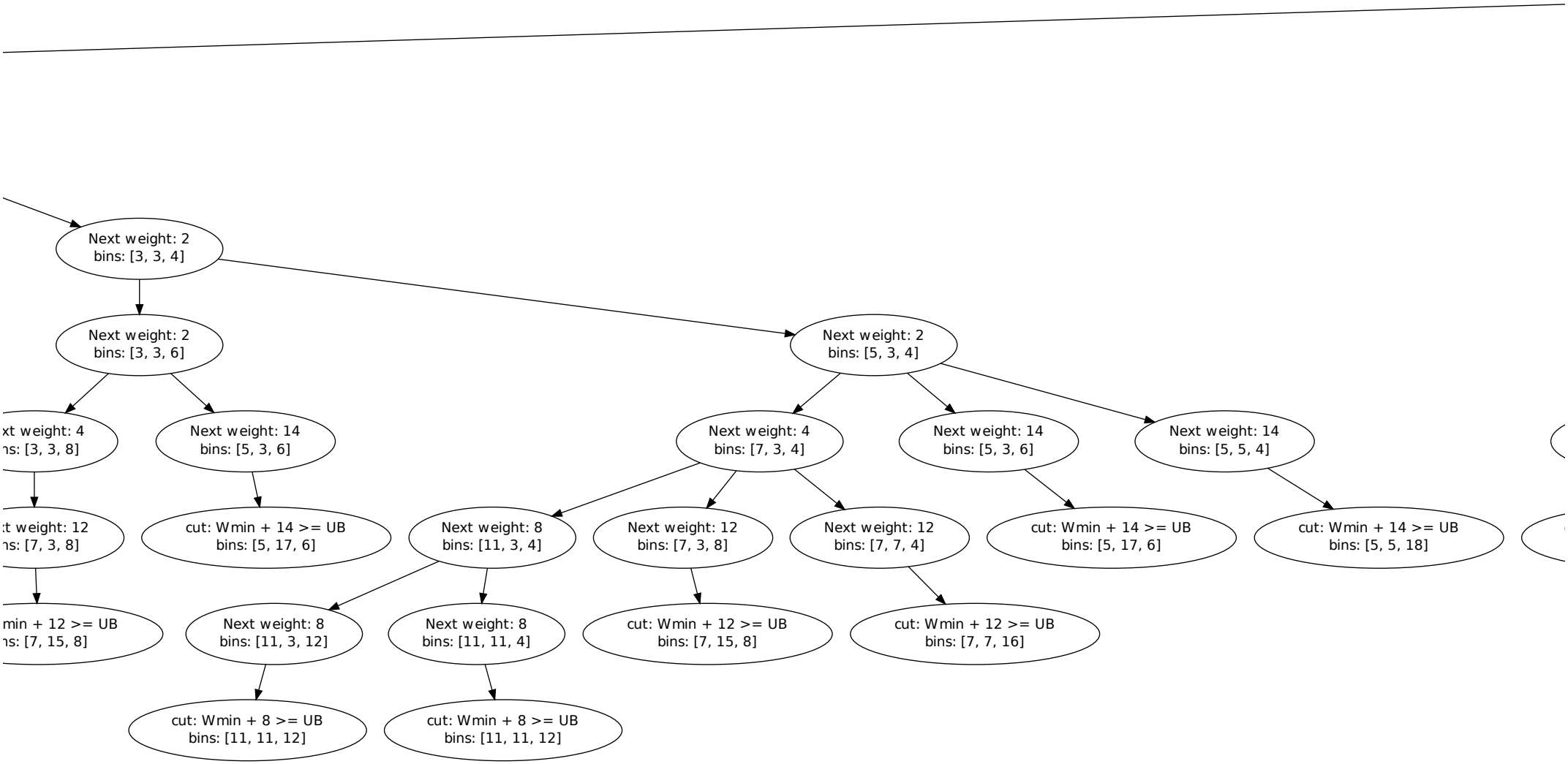

Next weight: 5
bins: [2, 2, 0]
Next weight: 3
bins: [7, 2, 0]
Next weight: 14
bins: [7, 5, 0]
Next weight: 4
bins: [7, 2, 3]
cut: Wmin + 14 >= UB
bins: [7, 5, 14]
Next weight: 9
bins: [11, 2, 3]
Next weight: 14
bins: [7, 2, 7]
Next weight: 13
bins: [7, 6, 3]
Next weight: 9
bins: [11, 2, 12]
Next weight: 9
bins: [11, 11, 3]
cut: Wmin + 12 >= UB
bins: [7, 16, 7]
cut: Wmin + 13 >= UB
bins: [7, 6, 16]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]
Next weight: 5
bins: [2, 2, 5]
Next weight: 10
bins: [2, 2, 10]
Next weight: 14
bins: [7, 2, 5]
Next weight: 9
bins: [12, 2, 10]
cut: Wmin + 14 >= UB
bins: [7, 16, 5]
cut: Wmin + 9 >= UB
bins: [12, 11, 10]
Next weight: 10
bins: [12, 1, 1]
Next weight: 10
bins: [12, 11, 1]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]

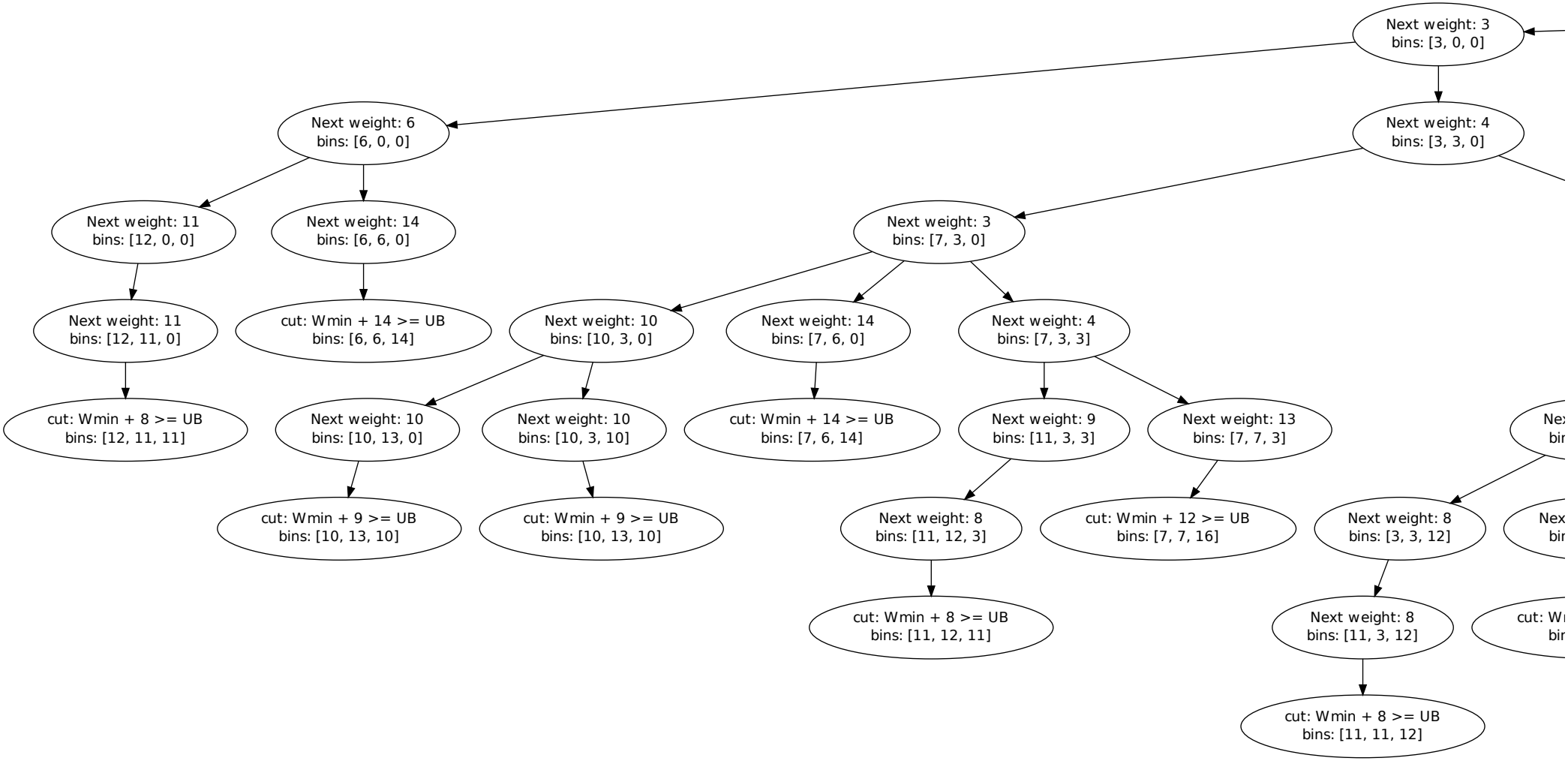

Next weight: 1
bins: [2, 1, 0]
Next weight: 2
bins: [3, 5, 2]
Next weight: 4
bins: [3, 7, 2]
Next weight: 14
bins: [5, 5, 2]
Next weight: 2
bins: [3, 5, 4]
Next weight: 10
bins: [10, 13, 0]
Next weight: 14
bins: [7, 7, 2]
Next weight: 13
bins: [3, 7, 6]
cut: Wmin + 14 >= UB
bins: [5, 5, 16]
Next weight: 4
bins: [3, 7, 4]
Next weight: 14
bins: [3, 5, 6]
Next weight: 14
bins: [5, 5, 4]
cut: Wmin + 9 >= UB
bins: [10, 13, 10]
cut: Wmin + 13 >= UB
bins: [16, 7, 6]
Next weight: 8
bins: [3, 11, 4]
Next weight: 12
bins: [3, 7, 8]
Next weight: 12
bins: [7, 7, 4]
cut: Wmin + 14 >= UB
bins: [17, 5, 6]
cut: Wmin + 14 >= UB
bins: [5, 5, 18]
Next weight: 8
bins: [3, 11, 12]
Next weight: 8
bins: [11, 11, 4]
cut: Wmin + 12 >= UB
bins: [15, 7, 8]
cut: Wmin + 12 >= UB
bins: [7, 7, 16]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]


Next weight: 4
bins: [3, 1, 2]
Next weight: 5
bins: [7, 1, 2]
Next weight: 5
bins: [3, 1, 6]
weight: 14
: [7, 1, 7]
Next weight: 14
bins: [7, 6, 2]
Next weight: 9
bins: [3, 1, 11]
Next weight: 14
bins: [8, 1, 6]
Next weight: 14
bins: [3, 6, 6]
+ 13 >= UB
', 15, 7]
cut: Wmin + 13 >= UB
bins: [7, 6, 16]
Next weight: 10
bins: [12, 1, 11]
Next weight: 9
bins: [3, 10, 11]
cut: Wmin + 13 >= UB
bins: [8, 15, 6]
cut: Wmin + 13 >= UB
bins: [17, 6, 6]
Next weight: 9
bins: [3, 11, 2]
cut: Wmin + 9 >= UB
bins: [12, 10, 11]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]
cut: Wmin + 9 >= UB
bins: [12, 10, 11]
Next weight: 9
bins: [12, 11, 2]
Next weight: 9
bins: [3, 11, 11]
cut: Wm
bins
cut: Wmin + 8 >= UB
bins: [12, 11, 11]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]


Next weight: 2
bins: [3, 1, 0]
Next weight: 4
bins: [3, 3, 0]
Next weight: 4
bins: [3, 3, 4]
Next weight: 4
bins: [3, 3, 8]
Next weight: 4
bins: [7, 3, 4]
Next weight: 9
bins: [12, 1, 2]
Next weight: 8
bins: [3, 3, 12]
Next weight: 12
bins: [7, 3, 8]
Next weight: 8
bins: [11, 3, 4]
Next weight: 12
bins: [7, 3, 8]
Next weight: 12
bins: [7, 7, 4]
Next weight: 10
bins: [12, 1, 11]
Next weight: 9
bins: [12, 10, 2]
cut: Wmin + 12 >= UB
bins: [7, 15, 8]
Next weight: 8
bins: [11, 3, 12]
Next weight: 8
bins: [11, 11, 4]
cut: Wmin + 12 >= UB
bins: [7, 15, 8]
cut: Wmin + 12 >= UB
bins: [7, 7, 16]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]


Name: Root
Next weight: 2
Next weight: 1
bins: [2, 0, 0]
Next weight: 6
bins: [5, 1, 0]
Next weight: 11
bins: [11, 1, 0]
Next weight: 14
bins: [5, 7, 0]
Next weight: 14
bins: [5, 1, 6]
Next weight: 3
bins: [7, 3, 0]
Next weight: 11
bins: [11, 12, 0]
Next weight: 11
bins: [11, 1, 11]
cut: Wmin + 14 >= UB
bins: [5, 7, 14]
cut: Wmin + 14 >= UB
bins: [5, 15, 6]
Next weight: 10
bins: [10, 3, 0]
Next weight: 14
bins: [7, 6, 0]
Next weight: 5
bins: [7, 3, 3]
cut: Wmin + 8 >= UB
bins: [11, 12, 11]
cut: Wmin + 8 >= UB
bins: [11, 12, 11]
Next weight: 10
bins: [10, 13, 0]
Next weight: 10
bins: [10, 3, 10]
cut: Wmin + 14 >= UB
bins: [7, 6, 14]
Next weight: 8
bins: [12, 3, 3]
cut: Wmin + 9 >= UB
bins: [10, 13, 10]
cut: Wmin + 9 >= UB
bins: [10, 13, 10]
Next weight: 8
bins: [12, 11, 3]
cut: Wmin + 12 >= UB
bins: [7, 8, 15]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]


Next weight: 2
bins: [3, 3, 4]
Next weight: 2
bins: [3, 3, 6]
Next weight: 2
bins: [5, 3, 4]
xt weight: 4
ns: [3, 3, 8]
Next weight: 14
bins: [5, 3, 6]
Next weight: 4
bins: [7, 3, 4]
Next weight: 14
bins: [5, 3, 6]
Next weight: 14
bins: [5, 5, 4]
t weight: 12
ns: [7, 3, 8]
cut: Wmin + 14 >= UB
bins: [5, 17, 6]
Next weight: 8
bins: [11, 3, 4]
Next weight: 12
bins: [7, 3, 8]
Next weight: 12
bins: [7, 7, 4]
cut: Wmin + 14 >= UB
bins: [5, 17, 6]
cut: Wmin + 14 >= UB
bins: [5, 5, 18]
min + 12 >= UB
ns: [7, 15, 8]
Next weight: 8
bins: [11, 3, 12]
Next weight: 8
bins: [11, 11, 4]
cut: Wmin + 12 >= UB
bins: [7, 15, 8]
cut: Wmin + 12 >= UB
bins: [7, 7, 16]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]


Next weight: 3
bins: [3, 0, 0]
Next weight: 6
bins: [6, 0, 0]
Next weight: 4
bins: [3, 3, 0]
Next weight: 11
bins: [12, 0, 0]
Next weight: 14
bins: [6, 6, 0]
Next weight: 3
bins: [7, 3, 0]
Next weight: 11
bins: [12, 11, 0]
cut: Wmin + 14 >= UB
bins: [6, 6, 14]
Next weight: 10
bins: [10, 3, 0]
Next weight: 14
bins: [7, 6, 0]
Next weight: 4
bins: [7, 3, 3]
cut: Wmin + 8 >= UB
bins: [12, 11, 11]
Next weight: 10
bins: [10, 13, 0]
Next weight: 10
bins: [10, 3, 10]
cut: Wmin + 14 >= UB
bins: [7, 6, 14]
Next weight: 9
bins: [11, 3, 3]
Next weight: 13
bins: [7, 7, 3]
cut: Wmin + 9 >= UB
bins: [10, 13, 10]
cut: Wmin + 9 >= UB
bins: [10, 13, 10]
Next weight: 8
bins: [11, 12, 3]
cut: Wmin + 12 >= UB
bins: [7, 7, 16]
Next weight: 8
bins: [3, 3, 12]
cut: Wmin + 8 >= UB
bins: [11, 12, 11]
Next weight: 8
bins: [11, 3, 12]
cut: Wmin + 8 >= UB
bins: [11, 11, 12]